Side Channels in the Cloud: Isolation Challenges, Attacks, and Countermeasures

Mohammad-Mahdi Bazm, Marc Lacoste, Mario Südholt, Jean-Marc Menaud

Side Channels in the Cloud: Isolation Challenges, Attacks, and Countermeasures

Mohammad-Mahdi Bazm*, Marc Lacoste*, Mario Südholt†, Jean-Marc Menaud†

*Orange Labs, France
†IMT Atlantique, France

***Abstract*—Cloud computing is based on the sharing of physical resources among several virtual machines through a virtualization layer providing software isolation. Despite advances in virtualization, data security and isolation guarantees remain important challenges for cloud providers. Some of the most prominent isolation violations come from side-channel attacks that aim at exploiting and using a leaky channel to obtain sensitive data such as encryption keys. Such channels may be created by vulnerable implementations of cryptographic algorithms, exploiting weaknesses of processor architectures or of resource sharing in the virtualization layer. In this paper, we provide a comprehensive survey of side-channel attacks (SCA) and mitigation techniques for virtualized environments, focusing on cache-based attacks. We review isolation challenges, attack classes and techniques. We also provide a layer-based taxonomy of applicable counter-measures, from the hardware to the application level, with an assessment of their effectiveness.**

***Index Terms*—side-channel attacks, cloud computing, cache-based side-channel attacks, timing attacks, isolation.**

I. Introduction

Cloud computing enables on-demand access to a shared pool of computing, storage and networking resources. This model allows customers to use mutualized software and hardware resources, abstracted as services, and hosted by cloud providers such as Amazon. Virtualization is the main enabler for sharing physical resources of a host among virtual machines through a hypervisor abstraction layer [1]. By partitioning physical resources, virtualization enables *software isolation* between virtual machines. However, resource sharing among different users and hardware virtualization are challenging [2] [3]. Many optimization mechanisms and techniques have therefore been proposed at the application, hypervisor, OS and hardware levels [4]. Unfortunately, despite their benefits, such approaches raise major security concerns, in particular, because resource sharing between several entities has a strong impact on isolation in such environments. Potential threats include data confidentiality and integrity, and availability of provided services (e.g., DDoS threats [5]). Among known threats to data security in a virtualized environment such as the cloud, *Side-Channel Attacks (SCA)* target highly sensitive data and computations, e.g., cryptographic operations. SCAs use a hidden channel that leaks information on an operation such as AES encryption, typically execution time or cache access patterns. Such channels are commonly created in software implementation of cryptographic algorithms, in a number of techniques and mechanisms and techniques widely used in hypervisors such as *memory deduplication* and in the hardware design itself, e.g., in the *Last-Level Cache (LLC)* of processors. SCAs can be applied to a wide range of computing devices from smartcards to VMs. Their impact may be greater than other attacks targeting cryptographic algorithms as they attempt to retrieve secret data without any special privileged access and in a non-exhaustive manner.

There are different categories of the attack regarding to the type of exploited channel; *Timing attacks*, *Cache attacks Electromagnetic attacks*, and *Power-monitoring attacks*. Electromagnetic attacks and power-monitoring attacks are more applicable to physical devices such as smartcards. Cache-based and timing attacks are the main software attacks applicable in cloud computing because of sharing the resources and virtualization techniques. To mitigate these attacks, countermeasures may be applied at three different levels: application, system, and hardware. Each level of defense has its own challenges, benefits and limitations w.r.t. metrics for assessing effectiveness, e.g., performance overhead, or compatibility with legacy. Application-level approaches to mitigation are based on programming and software implementation techniques to make sensitive applications such as OpenSSL SCA-resistant. Instead, system and hardware approaches explore how to enforce loose and strong isolation between computing elements respectively. While isolation risks and countermeasures are now fairly well-understood for a virtualized system when considering control of main communication channels (authorization), it is much less so when channels are indirect (side-channels). Due to the increasing complexity of virtualization stacks, blending multiple layers of variable levels of trustworthiness, it is thus important to provide a comprehensive overview of those security issues to help architects, developers, and security teams to prevent, detect, and react to such attacks by pinpointing the key elements not to overlook and by suggesting some countermeasures.

The objective of this paper is to provide an in-depth survey of security issues related to side-channels for virtualized systems. We identify security challenges, and cover possible attack types and existing countermeasures, providing also some

recommendations regarding possible mitigation techniques. We also sketch some directions for future attacks, extending towards distributed SCAs.

At the date of the writing this paper, we have found three surveys on side-channel attacks [6]–[8], of which only the latter addresses attacks in virtualized environments. The latter discusses threats and challenges in the cloud, provides some information on the cache architectures of the processors, a subset of current cache attack techniques in virtualized environments, as well as several countermeasures. These attacks are becoming a more and more important security concern in cloud computing. We present a taxonomy of attacks, and discuss types of attacks as well as countermeasures.

We provide the following original contributions :

- **Root cause analysis.** We identify the root causes for isolation breakouts in virtualization which can be exploited as SCAs.
- **Survey of attack techniques.** We provide a complete and detailed survey of attack types and attack techniques.
- **Survey of countermeasures.** We present a taxonomy of applicable countermeasures which are finely categorized into three broad kinds of mitigation, with a brief description of approaches and several security solutions for their root causes.
- **Security recommendations.** We provide security recommendations useful for mitigation of side-channel attacks, focusing on new virtualized infrastructures that have been targeted by recent attacks.

This survey is organized as follows. Section II provides some background on cloud computing and virtualization. Section III reviews security challenges for a virtualized environment, focusing on isolation. Sections IV and V explain side-channel attacks and their techniques. Section VI discusses approaches for attack mitigation at different levels. Section VII gives an outlook of future attacks, potentially concerned systems and infrastructures, and security recommendations for mitigation. Finally, Section VIII concludes the paper.

II. Cloud and Virtualization Overview

Three main cloud deployment models may be distinguished. A *private cloud* is owned and operated by a single company to provide for its IT requests, while a *public cloud* is operated by a cloud provider to provide IT services to, typically large numbers of, external customers. A *hybrid cloud* uses the base of a private cloud and combines it with the use strategy of public cloud services. Three types of services are customary offered by cloud providers. *SaaS* and *PaaS* constitute higher-level services, while *IaaS* are low-level services provided by a cloud provider. Security for *SaaS* and *PaaS* services is guaranteed by the cloud provider because customers do not have permission to access to underlying components of the service. However, IaaS security is a concern for the provider as the customer has complete control on the rented service.

Virtualization is the cornerstone of cloud computing by enabling multiple operating systems to be consolidated on a virtualization layer so that physical resources can be shared among them. This, in turn, allows for dynamic resource allocation and service provisioning. There are different types of virtualization; *paravirtualization*, *full-virtualization* and *hardware-assisted virtualization*. Using paravirtualization the guest OS is modified. It require substantial OS modifications in user applications. However, the hypervisor should support this capability. This technique is not fully transparent from a VM's point of view [1]. Using full virtualization, all system resources are virtualized *i.e.*, processors, memory, and I/O devices and run an unmodified operating system including all its installed software on top of the host operating system. This is a fully transparent technique. Hardware-level virtualization is nowadays supported by Intel and AMD directly on the processor level, respectively by VT-X and AMD-V techniques.

In contrast to VM-based virtualization techniques, Linux containers are often used as a lightweight virtualization solution. LXC and Docker containers that are designed to facilitate container management. Despite advantages of containers, they constitute, however, not the final word in lightweight virtualization. Recently, unikernels have been developed which are more flexible and reliable than containers. They are more lightweight than Docker containers and offer excellent security properties by reducing the attack surface through a thin virtualization layer. Regarding security, the choice between VM and container is a challenging decision. Securing containers is much more challenging than other virtualization methods, even though containers create a strong isolating environment for users. Cloud security frequently relies on hypervisors that implement security mechanisms such as isolation between virtual machines. Isolation is the main security-related feature in virtualized environments. Generally, isolation can be implemented at two different levels: at the *hardware* and *software* levels. As an example of hardware isolation, dedicating a L1 cache to each CPU core provides a rigid isolation between two cores of CPU. The sliding of a LLC in Intel processors is another kind of hardware isolation to protect per-core data. Software-based isolation provided by are usually implemented in the OS or at the virtual machine monitor (VMM) level to isolate two threads, processes or virtual machines.Varadarajan *et al* [9] suggest an approach to provide software isolation between two VMs at the hypervisor level. Co-residency — *i.e.*, different customers placed in the same physical infrastructure, VMs placed on the same physical machine — has an important security impact in such environments [10]. Embedded security mechanisms can then be bypassed by an attacker, thus threatening the security of data and computations. Side-channel attacks are an example of such attacks. Furthermore, in the case of containers, they execute on the same operating system. A host therefore shares its resources among containers which are owned by different users.

III. Security Challenges of Virtualization

Using virtualization, physical hosts can run several virtual machines in parallel, typically by means of a hypervisor. VMs are then executed on the cores of a processor. However, virtu-

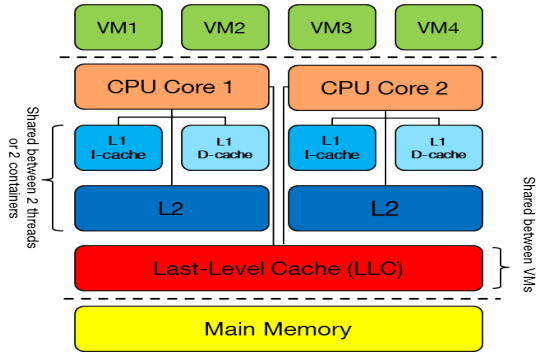

Figure 1: Sharing of resources and cache issue in virtualization

alization technology may have a strong impact on the security of hosts and hypervisor. In the following, we present several security challenges which may directly/indirectly (Table. II) compromise isolation in a virtualized environments (Table. I).

A. Issues of Shared Caches in Modern Processors

The memory hierarchy, especially cache memory, plays an important role for the performance of a computer system [11]. Caches are specialized memory layers placed between the CPU and the main memory (RAM). They are used to significantly improve the execution efficiency by reducing the speed mismatch between the CPU (which operates in GHz clock cycles) and RAM (whose accesses require hundreds of clock cycles). Today, there are typically three levels of cache: L1, L2 and L3 (LLC, last-level cache) levels. The L1 and L2 caches are usually private to each core and the LLC is shared among all cores of processor. The LLC is much larger (size of megabytes) than the L1 and L2 caches (which are sized in kilobytes). Caches are divided into equal blocks called *ways* and consisting of *cache lines*. Cache lines with the same index in ways form a *set* (Fig. 2). When the processor needs access (read or write data) to a specific location in RAM, it checks whether a copy of data is already present in the L1, L2 or LLC caches. If the data is found at any cache level (a *cache hit*), the CPU performs operations on the cache using only a few CPU cycles. Otherwise a *cache miss* has occurred and the processor needs to bring data from RAM at a much slower pace. Multi-core processors are nowadays widely used in a

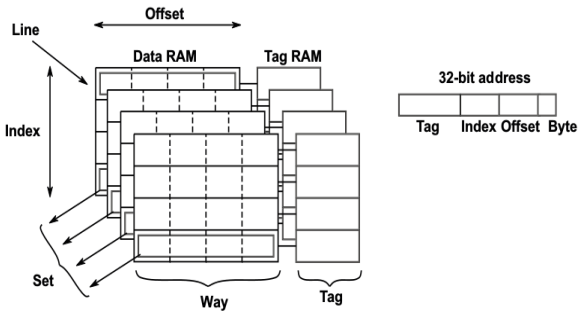

Figure 2: Cache Organization in CPU (source: [12])

virtualized environment because of their potential for powerful parallel processing. However, the cache hierarchy, especially the LLC, can be exploited as a vulnerable channel that leaks information on processes running on the cores. Figure 1 shows the cache hierarchy and its position w.r.t. VMs and the main memory.

B. Inclusive caches

Two cache architectures are common in modern processors: *inclusive* and *exclusive* ones. Exclusive caches do not use redundant copies. Exclusiveness, *e.g.*, used in the AMD Athlon processor, enables higher cache capacity. In *inclusive* architectures, all cache lines in L1 and L2 are also available in the LLC. For example, the architecture of LLC in Intel's Nehalem processors is inclusive in order to reduce snoop traffic between cores and sockets of the processor [13]. Inclusiveness provides less cache capacity but offers higher performance. On the other hand, inclusive caches may be exploited in a malicious way to perform cache-based side-channel attacks. When a cache line is evicted from the LLC, it is also evicted from L2 and L1 caches. A malicious user can then perform an attack by evicting caches lines from the LLC and measure the fetch time of a memory line from RAM, to gather tracking information on the fetched lines.

C. Simultaneous multi-threading

Simultaneous multi-threading (SMT), such as Intel's Hyper-Threading (HT), enables multiple threads to be run in parallel on a single core of a processor in order to improve execution speed. SMT allows sharing of dedicated resources of a core, mainly its L1 cache, between multiple threads running on the same core. This sharing can leak information between threads (of a core) and may be used by a malicious thread to gather information from other threads. SMT can potentially ease cache-based side-channel attacks [14] [15].

D. Data Deduplication

Online storage is one of the most-popular services offered by cloud providers. To maximize the number of users of a storage service, cloud providers constantly look for new means to store data more efficiently. *Data deduplication* is a widely-used technique to this end. It stores a single copy of redundant data belonging to different users. In addition, it also saves network bandwidth by requiring much less data transfers across network. However, data deduplication is subject to security issues because it can create a leaky channel between two users of the same cloud storage service [16].

E. Page Deduplication

Another widely-used optimization technique in operating systems and hypervisors is page deduplication to optimize memory usage [17]. Generally, an operating system loads libraries which are used by programs in the main memory. Then, related processes access to the same memory pages harnessing the mapping between its virtual addresses and the pages of the libraries. The most frequent form of page deduplication is *content-based* in which identical pages shared among

several processes are merged by the operating system or the hypervisor. It typically enables hypervisors to save memory, but also cache-based side-channel attacks [18] [19] [10] [20] [21]. Furthermore, mapping the virtual address space of two processes to the same physical pages can leak information. Consequently, it may be used as a method to exploit cache activities of a process or a VM.

F. Large Page Memory

The memory management unit (MMU) in hypervisors uses memory paging techniques to improve VM performance. One of these techniques is large page memory. Inside a processor, memory address translation is performed by translation lookaside buffers (TLB) and large page memory improves the efficiency of TLB accesses by reducing table entries. Default page size used in OS is 4KB, but the size of a large/huge page is 2MB or 1GB. Although this technique is useful in term of efficiency, it is also subject to severe security issues as part of the mapping of a virtual address to a physical one. Furthermore, large page memory divulges more bits of a physical address in the mapping process to an adversary than the small page memory. In fact, many bits of the virtual address remain unchanged in mapping process to physical address. Consequently, it gives more resolution to attack on the LLC that need selecting special sets [22].

G. Preemptive Scheduling

Using a time-based scheduling algorithm, schedulers divide CPU time into equal slices and allocate each slice to a virtual machine for executing its instructions on processor. However, the scheduler can interrupt the execution of a VM and allot CPU time to a priority task or VM. In this case, the switch time between two VMs can constitute a security risk. An attacker can force the scheduler to interrupt the execution of a victim VM. When a switch happens between the malicious VM and the victim VM, the attacker can get the state of the victim VM, *e.g.*, the victim's cache activities just after context-switching.

H. Non-Privileged access to hardware instructions

Almost all side-channel attack techniques exploit hardware instructions, such as `rdtsc` and `clflush`, to perform attacks. Furthermore, these instructions are used to measure time and to evict a memory line from cache, respectively. As the execution of such instructions is not controlled, they may be used by an untrusted process to manipulate CPU caches. To ensure the trusted execution of a process, Intel has introduced the *Software Guard Extensions* (SGX) [23]; in modern ARM processors, the execution of certain hardware instructions is forced to be privileged from the top layers.

Security Challenge	Description
Shared Cache in Processors	Sharing of CPU's cache between threads and VMs running on a processor
Inclusive caches	In an inclusive cache all memory lines in L1 are also present in L2 and LLC
Simultaneous multi-threading	Running multiple threads in parallel on a single core of,CPU
Data Deduplication	Merging of redundant data in order to optimize memory usage
Page Deduplication	Condensing redundant memory pages by hypervisor to obtain more RAM
Large Page Memory	Using large page memory to improve the efficiency of TLB access
Preemptive Scheduling	Time scheduling algorithm in which scheduler divides CPU's time to equal slices and attributes each one to threads
Non-Privileged access to instructions	Using hardware instructions in a malicious way to exploit important information

Table I: Identified challenges of virtualization technology regarding to the isolation issue

Layer	Mechnism / Technique	Direct	Indirect
Hardware	Cache architecture in multi-cores CPU	✓	
	Exclusive/inclusive cache		✓
	Simultaneous Multi-Threading (SMT)	✓	
Virtualization	Memory deduplication	✓	
	Large-Page memory managment		✓
	Preemptive scheduling		✓
	Non-privileged access to hardware instructions		✓

Table II: Impact of the techniques on the isolation in virtualized environments. If a technique plays a helper role in the attack steps, it has a indirect impact on isolation.

IV. Side-Channel Attacks

The use of cryptography is raising in many fields of computer science in recent years. Its applications abound especially in cloud computing for securing data and services [24] [25]. Since many cryptographic algorithms are documented and publically accessible, the security of a cryptographic system frequently relies on the robustness of its secret key. Among attacks which target a cryptographic cipher, side-channel attacks threaten the security of a system by allowing the encryption key to be inferred. Generally, each application has a especial behavier either in term of cache access or execution time. To perform certain types of the attacks, an attacker requires access to a VM running encryption operations. With cloud technology that is based on the virtualization, new means have became available to access to other guest operating systems which are hosted on a same physical machine. Cryptographic algorithms can be attacked more flexibly. Hence, the traditional implementations of these algorithms need to by verified in order to be compatible with virtualized environments [26].

A. Basic concepts: side channels, side channel attacks

In computer science, a *side-channel* is a hidden channel that is exists already in hardware, *e.g.*, in a processor's cache implementation, or is created using hardware and software techniques, employed *e.g.*, for improved resource utilization (hyper-threading, page deduplication, etc.). Any information obtained from such a channel is called side-channel information [7]. This channel is not a bidirectional channel and cannot be accessed directly by an adversary. A side-channel can be:

- *Inter-Process*: This category of channel is usually established between two processes, *i.e.*, a spy process and a victim process, that run in the same operating system space, typically on the same core. The spy process must be run in parallel with the victim process and both must be well synchronized.
- *Inter-VM*: This channel is either created between two VMs running on the same CPU core (core co-residency)

or on different cores of a CPU (cross-Core). The synchronization between two VMs running on the different cores is more difficult. In addition this type of channel has to cope with more noise than inter-process channels. Finally, a side-channel can be created through a network, yielding a more noisy channel because of network properties, such as the latency.

Any kind of attack that benefits from side-channel information is called a *side-channel attack*. A side-channel attack can be either *passive* or *active*. In passive attacks, the attacker observes activities of the target without performing any changes to the target in order to obtain information. In contrast, active attacks change the environment of the attack target by forcing the target to perform abnormal operations. Generally, three types of side-channels are distinguished: *CPU-based* ones, *cache-based* ones and *time-based ones*.

a) CPU Load-based side-channel: The CPU is one of the resources that is shared between multiple virtual machines. The CPU load can be used as a covert channel between two virtual machines that are running on a single CPU. Since different operations result in different CPU loads, these differences result in information leakage [27].

b) CPU Cache-based side-channel: Cache lines which are accessed during the encryption of plain-text passages, can reveal sufficient information to reconstruct an encryption key. Since the L1 and L2 caches are shared between processes running on a single core and the LLC is shared between all cores of processor [13] [28], a pattern of cache accesses by different processing entities (*e.g.*, processes and VMs) can be discovered in a malicious way. For example, as part of the software implementation of AES, lookup tables are pre-calculated and stored in RAM in order to improve performance. During encryption, the CPU loads the tables into the cache and the tables will be accessed by the encryption process. Traces of cache accesses disclose indixes into the tables to the attacker.

c) Time-based Side-Channel: The execution time of applications more especially cryptographic ones is not constant because it strongly depends on branches and the execution time of the algorithm's internal instructions. Therefore, different executions of an algorithm result in different execution times. Timing differences can thus also be used to leak information and can be exploited to retrieve secret data.

V. Timing and Cache-Based Side-Channels Attacks

Timing attacks and *cache-based attacks* are two main classes of side-channel attacks, respectively passive and active attacks. CPU-Load is used basically as a covert channel or an information flow between two virtual machines in a virtualized environment to bypass security policies. Table. III gives an overview of the corresponding security vulnerabilities and attacks. We now explain these attack classes in detail.

A. Timing Attacks

Time is often exploited as a leaky channel. An attacker may attempt, for example, to analyze the time taken to execute cryptographic algorithms in order to extract encryption keys. *Modular exponentiation* is an important operation widely used in the implementation of many cryptographic algorithms such as AES. The execution time of the frequently-used square-and-multiply method, *e.g.*,, is not constant and can be exploited to leak information through timing. Although such attacks are better applicable to limited computing devices such as smart cards, they can target also network-based cryptosystems and software implementations of cryptographic algorithms, such as `OpenSSL` and `Blowfish`. Timing attacks can be *local* or *remote*. As part of a local attack, a spy program is executed on the same machine as a victim program. In a remote attack, victim and attacker are hosted on different machines, in a local network or in the cloud. Remote attacks are notoriously difficult to tackle due to the noise induced by the network.

Such attacks were first proposed by *Kocher et al.* [39] who targeted Diffie-Hellman, RSA, and DSS implementations, notably on smart cards [40]. This kind of attack are also applicable to cloud-based cryptosystems that are distributed among several virtual machines. Brumley *et al*. [41] implemented a remote timing attack on RSA in `OpenSSL` in different environments between two processes, virtual machines and hosts in a local network, the efficiency of which was later improved [42]. *Cache-based timing attacks* [26], [43]–[45] leverage cache memory as a means to locally attack a cryptosystem. An example of such an attack was proposed for DES by inferring S-box inputs [46] by measuring the encryption time for different plaintexts. The execution time of an encryption algorithm may also vary due to its memory activity, *i.e.*, cache accesses. Cache misses notably increase the execution time of an algorithm at run-time. Following a similar approach than that by Tsunoo *et al* [45], [46], an attack against AES has been proposed that also uses S-boxes [43]: this attack is based on cache-collisions, that may be detected by a cache hit (taking less time than a cache miss). This class of timing attack does not manipulate directly cache memory, unlike cache-based ones, see below.

B. Cache-Based Side-Channel Attacks

A shared cache can also leak information. Cache-based attacks may be classified according to the type of side-channel along with different attack and helper techniques used to exploit cache.

1) Side-Channel Types: Two broad classes of channel-based attacks can be distinguished.

a) Access-Driven Attacks: The attacker tries to find any relation between an encryption process and accessed cache lines [15], [32], [34], [36]. Furthermore, the attacker exploits the pattern of cache accesses by the victim. To derive a profile of cache activities, the attacker manipulates cache by evicting memory lines of victim process. Such attacks are easier to perform than time-driven attacks, because they are less dependent on precise timing information.

b) Trace-Driven Attacks: The attacker observes cache activities, *i.e.*, memory lines which are accessed by an encryption process during its execution to obtain a sequence of

Paper	Vulnerability	Technique	Type	Target Program	Environment	Cache Level	Micro-architecture
Ristenpart et al. [29]	Shared cache	Prime+Probe	Access-driven	User activity	Amazon EC2	L1,L2	AMD Opteron
Irazoqui et al. [30]	Huge Page	Prime+Probe	Access-Driven	AES	Xen 4.1	LLC	Intel i5-650
Inci et al. [31]	Huge Page, Inclusive Cache	Prime+Probe	Access-Driven	RSA	Amazon EC2	LLC	Intel Xeon E5-2670
Gullasch et al. [32]	Shared cache	Prime+Probe	Access-Driven	AES	Arch Linux	L1	Intel Pentium M
Oren et al. [33]	Inclusive cache	Prime+Probe	Access-Driven	Tracking user behavior	Linux,Mac OSX 10.10	LLC	Intel Haswells,(Sandy&Ivy)Bridge
Yarom et al. [34]	Page sharing, Inclusive Cache	Flush+Reload	Access-Driven	RSA	VMware ESXi 5.1,KVM	LLC	Intel i5-3470,Xeon E5-2430
Gruss et al. [20]	Page Deduplication	Timing difference	Time-Driven	User activity	KVM,win8,Android	L1, LLC	-
Liu et al. [35]	Huge Page	Prime+Probe	Access-Driven	Elgamal	Xen4.4,VMware ESXi 5.1	LLC	Intel Xeon E5-2670,Intel i5-3470
Gruss et al. [36]	Page Sharing	Flush+Flush	Access-Driven	AES	Linux	LLC	Haswell i7-4790
Zhang et al. [37]	Page Sharing	Flush+Reload	Access-Driven	User shopping basket	Amazon EC2/Ubuntu/LXC	LLC	Intel Xeon E5-2665
Spreitzer et al. [38]	Disaligned AES T-tables	Evict+Time	Access-Driven	AES	Google Nexsus S	L1	Cortex-A8

Table III: List of cache-based side-channel attacks

cache hits and *cache misses* [47], [48]. For instance, observing which memory accesses to a lookup table lead to *cache hits* allows disclosing indices in the lookup table. After capturing enough samples an offline phase permits to infer the secret data that is used by the victim process.

2) Attack Techniques: Caches-based attacks have been performed through programs that have been implemented in languages such as C/C++ [18], [34] or in JavaScript [20]. The implementation methods vary because of differences in how the languages allow caches to be accessed. Different attack techniques have been proposed: *prime+probe*, *flush+reload*, *flush+flush*, *evict+time*.

a) Prime+Probe: This access-driven attack is commonly applied by an attacker to spy on the behavior of a cache shared between the attacker and a victim (VM or process). The attacker monitors which cache sets are accessed by the victim. This technique usually targets the L1 cache that is less noisy than the other cache levels. To perform such an attack, the attacker needs timing information (usually provided by hardware instructions such as `RDTSC`). The attack is performed in three main steps: 1) the attacker fills one or several selected cache sets – he selects more likely cache sets to be accessed by the victim; 2) the attacker waits for a defined period while the cache is used by the victim process; 3) the attacker refills the memory sets with the same data used in step 1 to verify which cache lines are accessed by the victim. Indeed, if a memory line was evicted by the victim's activities, re-accessing it requires the CPU to bring back the line from main memory which requires noticeably more CPU cycles. Otherwise, if the line is in the cache, *i.e.*, no need to fetch it from main memory, fewer CPU cycles are needed. Exploiting the time difference between a cache hit and a cache miss, an attacker can infer which lines of a cache set are evicted by the victim (see Fig. 3). *Ristenpart et al.* performed the first attack using this technique on the Amazon EC2 cloud service [29].

b) Flush+Reload: Many approaches use this technique to exploit the cache [32], [34], [37], [49]–[51]. like in Prime+Probe, an attacker needs to run a spy process inspecting memory lines accessed by the victim process. The attacker monitors which memory lines are evicted by the victim process from all the cache levels. Thus, it determines which memory lines were accessed by the victim, builds a profile of the victim cache activities (cache access patterns),

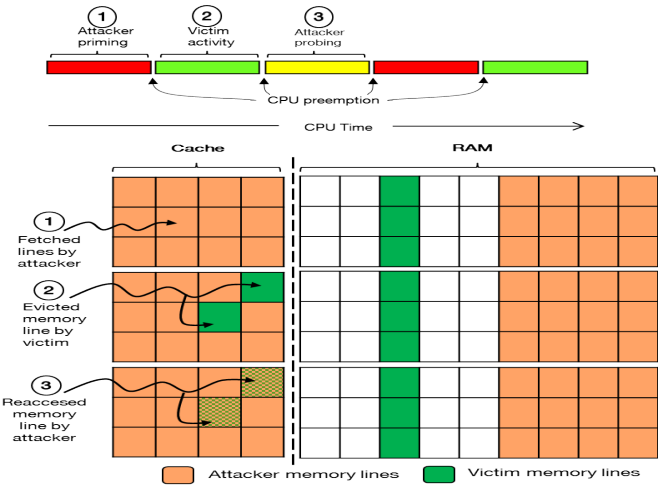

Figure 3: Three phases of Prime+Probe technique

and infers useful information from the data processed by the victim. This attack consists of three main phases. In the *flush* phase, the attacker flushes memory lines of the victim process from the cache. In the *Wait* phase, he pauses for memory lines to be re-accessed by the victim process. The duration of the *wait* phase is important and must be precisely chosen. Otherwise, the attacker is unable to capture the victim's cache activities. In the *reload* phase, the attacker reloads the memory lines that were evicted from the cache to determine which lines are accessed by the victim process [32]. To perform the attack without any knowledge of cache line addresses, the attacker and the victim must share the same virtual address space, and the hierarchy of cache must be inclusive [34]. This technique does not require any information on the virtual-to-physical address mapping to infer *physical addresses* of memory lines. Thus, virtual address space randomization techniques such as ASLR[1] are not effective to mitigate such attacks.

c) Flush+Flush: This variant of *Flush+Reload* was proposed by *Gruss et al.* [36], based on measuring execution time of the `clflush` instruction that evicts a memory line

[1]Address Space Layout Randomization

from the cache hierarchy. Unlike Flush+Reload where the spy process reloads data from main memory during the *Reload* phase, in Flush+Flush there is no memory access by the spy process. This technique is thus faster than Flush+Reload.

d) Evict+Time: This class of attack works quite differently than other techniques [38]. First, the attacker triggers the encryption of a *plaintext*, which causes the loading of memory pages which contain, *e.g.*, lookup tables into the cache. Usually, all tables are cached at this stage by the victim process. Then, the spy process accesses some memory pages in its own memory address space. This causes loading pages to the cache, thus evicting from the cache the tables loaded during the first phase. Finally, the attacker triggers the encryption of the same initial plaintext, and measures the encryption time. During encryption, prefetching a memory page in the cache takes more times than if the page is not re-fetched. This timing difference is used to infer a cryptographic key. This technique requires knowledge of memory addresses of the victim process and mapped addresses of memory sets in the cache. For example, in the case of the attack on AES, the attacker needs to know the virtual memory addresses of AES tables and which cache sets are mapped to those addresses [52]. This technique also requires accurate timing information.

e) Evict+Reload: As seen before, unlike Intel processors, the `clflush` instruction is not available to non-privileged users in certain processors *e.g.*, ARM. Therefore, to overcome this challenge, *Gruss et al.* [53] suggested provoking cache contention by replacing the memory through loading something else into the cache instead of using `clflush` instruction in the Flush+Reload technique. However, the efficiency of such a technique depends on the cache replacement policy.

f) Prime+Abort: This attack technique [54] exploits an integrated extension in Intel processors called *Intel TSX* that adds hardware transactional memory support, for profiling victim's cache accesses and consequently retrieve sensitive information. Unlike other attack techniques, this technique does not need precise timing information such as `rdtsc` to detect a cache miss/hit. This is why this technique is more flexible than other ones. Furthermore, to trace cache accesses done by the victim, the attacker takes advantage of two mechanisms present in *transactional* memories. In such memories, a *transaction* (*e.g.*, access to a cache line) is either *completed* or *aborted*. In the case of side-channel attacks, the attacker exploits abort signalization to see if a cache line is accessed by the victim. The attacker creates an eviction set to target a cache set like in the first phase of Prime+Probe technique through the TSX transaction. Then, he waits cache lines in the target cache set be accessed by the victim. If the victim accesses one of cache lines during its execution, an abort is triggered and the attacker sees this abort signal indicating that the victim is accessing to targeted cache lines. In such way, the attacker creates a profile of cache accesses of the victim without any timing information.

3) Attack Challengies:

a) Co-Residency with Victim VM: A side-channel attack requires at least two virtual machines (the malicious and victim ones) residing on the same physical host. In a cross-VM attack, an adversary should be able to place the malicious VM on the physical host of the victim VM. Therefor, the adversary needs to evaluate the behavior of cloud's placement algorithm in different scenarios and then use network tools such as *nmap* and *hping* to check co-residency based on network information [29]. In [55], they did several experiments to find vulnerabilities in placement policies of public clouds like Amazon EC2, Google Compute Engine and Microsoft Azure. The study shows that we can meet co-residency is these clouds by understanding the behavier of VM placement algorithm according several effective variables such as launching time of instance, number of instances and requested data center.

4) Helper Techniques for Cache-Based Attacks: Besides the main attack techniques, a number of other techniques are necessary for an adversary to improve the attack resolution and to reduce noise, especially when exploiting LLC. To this end, reverse engineering techniques, network tools as well as machine learning and filtering techniques have been employed.

a) Reverse Engineering of Cache Addressing: The LLC cache is large and shared between several cores: to improve CPU efficiency, it is split into *slices*, with one slice per core. Slice addressing schemes are complex and based on a hash function (*complex addressing function*) that maps memory lines to LLC slices. Hash function complexifies cross-core cache-based side-channel attacks. An attacker can uses reverse engineering to predict the slices that are used by a physical address. Furthermore, finding slices of an address in the LLC can help the attacker because of the size of the LLC. Therefore the attacker does not need to probe all memory cache sets. Several techniques were developed for reverse engineering the hash function [56]–[59]. Reverse engineering has also been used by Irazoqui *et al* [30] to exploit the LLC. In contrast, the attack by Yarom *et al* [34] does not need to discover used slices by address by using memory sharing. In [35], this limitation was bypassed by creating an eviction set using large pages.

b) Machine Learning and Filtering Techniques: In a cache-based attack, collected traces of cache activities are not necessarily clean because of the noise induced in the cache by other processes concurrently running on the same host. To extract the traces of the victim, an adversary needs to use filtering tools. For instance, the Fourier transform method has been used to eliminate the cache background noise [33]. Machine-learning methods such as clustering and classification may also be applied on the collected traces to extract clean information. For instance, *SVM* was used as a machine-learning technique to analyze the collected information from the L1 cache [60].

VI. Countermeasures

Mitigation techniques against timing and cache-based attacks may be divided into three broad classes according to the

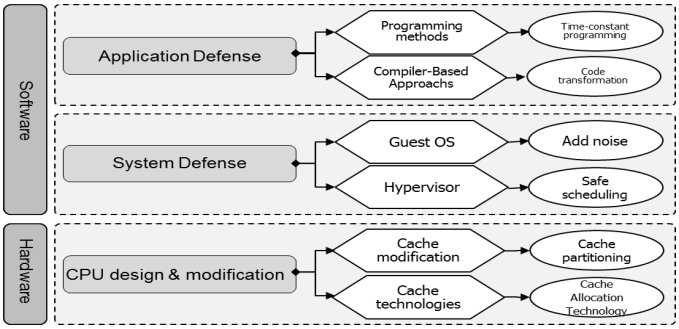

Figure 4: Taxonomy of defense approaches according to different layer of application

applicable enforcement layer in infrastructures: *application*, *system*, and *hardware* (Fig. 4). Many solutions only protect against a subset of attacks. Nearly all of them impose significant performance overheads. The ideal countermeasure should provide robust security with minimal performance degradation, while being scalable, and applicable to existing infrastructures with little modifications. Table. IV gives an overview of main approaches, corresponding mitigated attacks, and approach limitations.

A. Application-Based Approaches

A wide stream of research explores improving cryptosystem security through secure design and implementation at the application-level. Design-oriented solutions enhance SCA resistance of ciphers. Development and code optimization-oriented solutions propose new programming methods for secure implementation of cryptographic algorithms.

Robust and constant-time algorithms can eliminate SCAs, notably timing SCAs. A constant-time implementation should use instructions or operations with a fixed execution time. However, data-dependent branches of an implementation can alter execution time significantly. Correlations may be avoided by eliminating conditional branches and loops dependent on secret input data. [61]–[64] propose making constant-time the mathematical operations used in algorithms of common cryptosystems. *Masking cryptographic algorithm inputs* is also possible through randomization techniques [52], [65]–[67]. *Time padding* is another technique for timing SCA mitigation by introducing a delay in a function to hide timing leaks [68]. *Code transformation* for software protection is another approach to mitigate timing SCAs [69], notably through compiler-based techniques [70], [71]. Key-dependent control flow and dataflow might be the root cause of timing variations and correlations between secret data and execution time. Therefore, the compiler may be modified to eliminate branches to cut any such correlation using an *if-conversion* transformation [2] [71]. Timing variations related to dataflow properties of x86 processor pipelines were notably explored, with proposition of several compiler-based mitigations. Unfortunately, the overhead of such techniques remains high [70]. *Obfuscation* of cache access patterns from secret data may also be used to mitigate cache-based SCAs [44], [52]. Indeed, cache access patterns leak information. Thus memory access should not depend on secret information. Large lookup tables in a cipher implementation create data-dependent memory accesses, increasing effectiveness of all attacks [52], [72]. For example, for AES, permuting S-Box tables enables to obfuscate cache-access patterns for those tables, thus reducing attack vulnerability [73]. Programs may also be obfuscated at source code level [74]. A code transformation approach satisfying a *Memory Trace Obliviousness (MTO)* property was notably proposed [75]: memory access traces are hidden from the control flow of program to prevent any information leakage such as access pattern and access time.

[2]This code transformation technique converts control dependencies into data dependencies.

B. System-Based Approaches

1) System Level Counter-Measures: Mitigation at operating system and/or hypervisor levels might be the best choice, being applicable to existing cloud infrastructures with little modifications. For instance, an application-independent approach was proposed to prevent timing attacks [78]: it modifies the underlying OS combining *time-padding*, *cache cleansing*, and *dynamic partitioning* methods. Time-padding ensures that the execution time of a protected function is independent of the function input secret data. Padding thus prevents an attacker from measuring the execution time of the function. Cache cleansing prevents obtaining the state of the cache after running the sensitive function. Finally, cache partitioning allows protecting resources of a trusted process from being accessed by an untrusted process during its execution. Zang et al. [76] proposed *Düppel*; a system to mitigate cache-based timing attacks especially PRIME+PROBE on L1 and L2 caches. The approach consists in introducing some noise between PRIME and PROBE phases of the attack to reduce resolution of the attack. The system kernel does not need to be modified. This countermeasure can be easily deployed by installing a new kernel process. The overhead remains minimal (at most 7%). The *OS-level scheduler* is another possible SCA mitigation approach. Current system schedulers use time as index for context-switching among threads. An *instruction-based* scheduler was notably proposed to eliminate cache-based timing SCAs [79]. This approach schedules threads according to the number of instructions they execute. Processor performance monitoring units (PMUs) [95] enable to obtain the number of instructions executed through provided performance counters. *Cache line locking* is another OS-level mechanism to prevent cache-based SCAs in the cloud. It was suggested and implemented by Kim et al. [88]. The approach is based on the idea of locking and multiplexing cache lines for each VM using a software method. The software locks pages of a VM into the shared cache, a set of cache lines being locked and attributed to each CPU core. When a VM is running on a core, the VM pages are loaded to the locked lines and cannot be evicted by another VM running on other

Level	Approach	Target	Description	Drawback
Application	Constant-time implementation [61] [62] [63] [64]	Timing attacks	Writing constant-time algorithms	Very hard to write such programs
	Masking input data [65] [66] [67] [52]	Timing attacks	Randomization technique to obfuscate input-data	Complex implementation
	Time padding [68]	Timing attacks	Adding time as noise in functions	Performance degradation
	Code transformation [70] [71] [74]	Timing attacks	Using compiler to eliminate data and control depandancy from program	High performance overhead
	Obfuscating cache accesses [73]	Cache-based attacks	Eliminate cache accesses pattern from secret data	Complex implementation
OS	Adding noise [76]	Cache-based attacks	Adding noise into cache enter context-switching	High performance overhead
	Flushing cache [77] [78]	Cache-based attacks	flush cache lines from cache to clear access traces	High Performance overhead
	Instruction-scheduler based [79]	Cache-based attacks	Context-switching according to the number of instructions	Not supported by all CPUs
Hypervisor	Time-scheduler based [9]	Cache-based attacks	Play with scheduler's timer	Performance overhead
	Moving Target Defense [80] [81] [82]	Cache-based attacks	Leveraging VM placement algorithm to interrupt attacker VM	Performance overhead of VM migration
	Fuzzy time [83]	Cache-based attacks	Degrade RDTSC resolution	Affect time sensitive applications
	Static-page coloring [84] [85]	Cache-based attacks	Split cache to different areas statically	Performance overhead, Limited number of running thread
	Dynamic-page coloring [86] [87]	Cache-based attacks	Split cache to different areas dynamically	Complex implementation
	Locking cache lines [88]	Cache-based attacks	Lock cache lines by hypervisor	Performance overhead
Hardware	Random permutation [89] [90] [91] [92]	Cache-ased attacks	Random attributing of cache lines to threads	Performance overhead, Legacy
	Dynamic cache partitioning [93]	Cache-based attacks	Split cache to different areas dynamically	Complex implementation, Legacy
	Static cache partitionig	Cache-based attacks	Split cache to different areas statically	Performance overhead, Legacy
	Locking cache lines [94] [89]	Cache-based attacks	Lock cache lines during execution of threads	Performance overhead, Legacy

Table IV: An overview of suggested countermeasures according to each level of application

CPU cores. Multiplexing of cache lines allows obfuscating cache access patterns of a VM from co-resident VMs. Despite its small system overhead (2-5%), this approach requires to modify the guest operating system.

2) Hypervisor-Based Approaches: Countermeasures applied at this level could be more interesting for SCA mitigation than OS-level mechanisms as the cloud provider does not need to modify the guest OS of customers. For instance, a *scheduler-based approach* may use scheduler parameters to prevent cross-VM SCAs [9]: a scheduler parameter in Xen and KVM called `ratelimit_us` is used to interrupt a malicious VM tracing a victim VM. This parameter determines the minimum run-time of a virtual CPU (VCPU) on a physical CPU. The approach is simple to implement, but may degrade the efficiency of the hypervisor.

As another mitigation approach, *Moving Target Defense* (MTD) [81] is based on changing system configuration to make the attack surface dynamic and consequently harder to exploit by attackers. For instance, the cloud scheduler may be leveraged for this purpose. In fact, the scheduler may randomly decide to allocate the instance to a host in the infrastructure according to the instance placement policy to avoid co-residency between the malicious and victim VMs on the same physical machine. On the other hand, instance migration [82] may also be leveraged to mitigate co-residency attacks as a reactive approach. Such a mitigation approach requires support for detecting malicious VM and then react by migrating the malicious VM to another host in the cloud infrastructure. Actually, almost all the actual detection approaches take advantage of Hardware Performance Counters (HPCs) in different ways. HPCs provide high resolution information to detect such fine-grained attacks. *Zhang et al.* [96] presented a real-time detection framework to detect side-channel attacks conducted between two VMs in the cloud. Furthermore, they generate a signature for the sensitive application running in the victim VM, through different events provided by HPCs. The signature is then used by the detector to identify when the sensitive application is running. Once the application is identified, the detector starts to analyze performance events (*e.g.*, cache-misses) of the adversary VM to detect any malicious behavior through anomaly detection.

We already seen (cf. Section III-H), some hardware instructions are widely used to implement SCAs. The *Fuzzy time* approach prevents a VM from obtaining precise timing information, and may help to mitigate cross-VM SCAs in a IaaS system [83]. The idea is to degrade timing information provided by `RDTSC` instruction in a Xen-based virtualization environment. This hampers SCAs where fine-grained timing is needed to detect cache hits/misses. *Cache flushing* is another technique that eliminates any overlapping access to the cache between victim VM and adversary VM. Flushing creates strong data isolation between two VMs, i.e., each one only seeing its own data in cache. This server-side approach was notably implemented in Xen to prevent Prime+Probe cache-based SCAs for the cloud [77]. However, it suffers from performance overheads. *Page coloring* is an another software technique to mitigate cache-based attacks. A specific color (secure color) is assigned to pages of each VM. Pages with the

same color are then mapped to a fixed set of cache lines, only accessible to the related VM. Page coloring is done either *statically* [84], [85] or *dynamically* [86], [87]. Static page coloring degrades the performance of virtualized environment and limits the number of running VMs. In dynamic page coloring, cache protection mechanisms are only active during while running sensitive operations to improve the performance.

C. Hardware-Based Approaches

Hardware-level mitigation provides strong isolation between processing units. Proposed approaches include modifying cache architecture and integrating new hardware technologies, at cache-level or in terms of cryptographic processing.

1) New Cache Designs: This class of countermeasure proposes new designs (or modifications) for the cache architecture in processors. This approach provides strict isolation, with trade-offs to be found between security and degradation of cache performance. Proposed designs are mainly about *the cache replacement policy*, *locking cache lines*, or *partitioning the cache*. A new random replacement cache algorithm called *security-aware* was for instance proposed [91], [92]. A new cache architecture that dynamically locks cache lines used by a thread was also explored to protect against access-driven SCAs [94]. The allocated cache lines cannot be flushed from the cache by any other thread. The *Partition-locked cache* (PLCACHE) architecture locks cache lines, while the *Random-permutation cache* (RPCACHE) one eliminates any interference that may leak information from the cache [89]. Designs improving security of PLCACHE and RPCACHE caches were also explored in [90]. Cache partitioning was also investigated to mitigate cache-based SCAs, either statically or dynamically.

2) Hardware-Based Cache Technologies: Recently, Intel introduced the new *Cache Allocation Technology (CAT)* [97] in its processors to improve the performance of latency-sensitive applications by guaranteeing cache capacity to priority applications. CAT allows dynamic partitioning of the LLC between different CPU cores. Different classes of service are defined that can be assigned to each core. Thus, a portion (subset of cache lines) of the LLC will be assigned to selected cores, cache lines in each portion being only accessible by the assigned core. CAT was notably leveraged to mitigate SCAs on the LLC [98]: the LLC was divided into secure and non-secure partitions using CAT. Thus, the pages of sensitive code loaded into the secure partitions cannot be evicted by other VMs running on CPU.

3) Hardware-Implementation of Cryptographic Cipher: Hardware cipher embedding improves efficiency and security of a cryptographic algorithm. Intel introduced a new set of instructions in Xeon 5600 processors called *Intel Advanced Encryption Standard New Instructions* (Intel AES-NI) to accelerate AES encryption/decryption [99]. With AES lookup tables embedded in the processor and block encryption handled in hardware, this mechanism can be a secure solution for developers using AES in their programs, providing SCA-mitigation, notably for cache-based SCAs. However, this mechanism remains focused on AES, and is not available on all processors.

VII. Discussion

We previously identified a number of *security challenges* for virtualized environments, discussed in Section III. In terms of threats to isolation, root causes come from techniques which mostly aim to improve the performance of a virtualized environment. However, such techniques have a security impact on the environment in which they are applied (see Table. I).

A wide literature has already been published on different classes of attacks exploiting the previous security challenges to obtain sensitive information such as cryptographic keys. Some of them are shown in Table. III, with their mapping to the security challenges. From this big picture, we may observe that: (1) side-channel attacks can be applied to a large variety of OS and hypervisors; (2) the LLC is the most exploited shared resource in processors especially Intel ones. We also think that side-channel attacks may target any type of system, either a standalone or a distributed system composed of several sub-systems with different operating systems running in VMs which are consolidated on physical machines equiped with multi-core processors and the inclusive cache architecture.

In what follows, we propose a number of potential solutions for the previous security challenges for virtualized environments (Table. V). We also sketch how SCAs could be extended, either in terms of forms of attacks, or of cloud architectures. Finally, we conclude with some practical recommendations to reduce SCA occurrence in the cloud computing.

A. Extended Forms of Attacks: Distributed SCAs

From Table.III, we may observe that all attacks are for the moment performed on a single physical host, considering a single victim. Such attacks could also threaten the security of distributed systems, where all components of the system are hardened by cryptographic algorithms. More precisely, by applying several SCA attacks on the system components, an attacker could steal sensitive information from each component, and compromise the security of the overall distributed system. We believe that, in the future, novel SCA attacks will be performed in a distributed manner. Such distributed attacks could for instance be performed like *DoS* attacks on the memory of physical machines in the cloud [100]. These attacks may concern distributed computation such as secure multi-party computation [101] and multi-party machines learning approaches [102] based on homorphic cryptography to preserve privacy and security.

B. Extended Vulnerable Infrastructures: SCAs for Decentralized Cloud Infrastructures

To offer IT services to customers, cloud providers need to investigate how to build efficient IT infrastructures. Unfortunately, data center building and maintainance remains very expensive, and its capacity limited by physical resources *e.g.*, computing, storage and networking. Offered services should also be reliable in terms of response and processing time to satisfy customers. New architectures and paradigms for Decentralized Cloud Infrastructures (DCI) are emerging to solve some of those challenges [103]. For instance, *Edge Computing*

deports virtualized resources to the edge of networks. This class of architectures has many benefits such as improving response time and cloud efficiency by processing useful data in core data centers, and less important data at the edge. Although, traditional and distributed clouds differ in terms of architecture, they are still both based on virtualization technologies. Thus, SCAs could also threaten isolation for decentralized cloud infrastructures. However, the co-residency challenge remains as main challenge to overcome in the first step of side-channel attacks. This issue gets more attention in DCI because this kind of cloud composed of several self-managed data centers with different VM placement policies.

C. Security Recommendations

We formulate some security recommendations to cloud providers to avoid SCAs, especially those that exploit the processor.

- *Deactivate the SMT feature* to remove any shared path between two running threads on the same core of CPU.
- *Do not use, if possible, memory deduplication*, although this technique really increases memory performance.
- *Use processors with exclusive cache if possible*. As already seen, inclusive caches are widely exploited by hackers to perform SCAs. Processors with exclusive cache may effectively reduce the possibility of attacks such as *Flush+Reload*.
- *Alternatively, host all VMs owned by a user on a physical server*. By limiting multi-tenancy, this solution reduces the risk of isolation threat, and thus the probability of SCA occurrence.
- *Activating ASLR* makes these attacks more complicated, especially Prime+Probe.
- *Intel SGX* does not protect against side-channel attacks [104] because in such attacks, an attacker does not need to the content of memory lines that are protected by Intel SGX. In fact, cache access traces of victim are exploited, not memory contents.

Security Challenge	Solution(s)
Shared Cache in Modern Processors	Cache partitioning, Locking cache lines, Random permutation of cache lines
Inclusive cache	Limiting access to some instructions like `clflush`
Simultaneous Multi-Threading	Disable SMT, Locking cache lines, Page coloring
Page Deduplication	Obfuscation cache accesses in code, Degrade the resolution of timing instructions, Adding noise
Large Page Memory	Cache partitioning, Locking cache lines
Preemptive Scheduling	Safe scheduling, Adding noise, Flushing cache in context-switching
Non-Privileged access to instructions	Degrade the resolution of instructions, Limiting access to some instructions

Table V: Solutions for the security challenges in Table. I

VIII. Conclusion

In this paper, we have surveyed the state-of-the-art in side-channel attacks in cloud computing, its essential types, different techniques and several countermeasures according to different levels of application. These attacks can threat the security of cryptographic algorithms in any computing environment. This is why it is important to investigate in this field. Actually, there are many types of research on the improvement of attacks, adapting attacks to virtualized environments such as cloud and finding new attack techniques. On the other hand, there are many types of research on countermeasures to mitigate these attacks at different levels. However, we have noticed that there are not many mitigating approaches at the hypervisor level. As a sophisticated mitigation level, it needs to more investigations.

References

[1] J. Sahoo, S. Mohapatra, and R. Lath, "Virtualization: A survey on concepts, taxonomy and associated security issues," in *Computer and Network Technology (ICCNT), 2010 Second International Conference on*, pp. 222–226, IEEE, 2010.

[2] A. Menon, A. L. Cox, and W. Zwaenepoel, "Optimizing network virtualization in xen," in *USENIX annual technical conference*, no. LABOS-CONF-2006-003, 2006.

[3] R. Uhlig, G. Neiger, D. Rodgers, A. L. Santoni, F. C. Martins, A. V. Anderson, S. M. Bennett, A. Kagi, F. H. Leung, and L. Smith, "Intel virtualization technology," *Computer*, vol. 38, no. 5, pp. 48–56, 2005.

[4] G. Neiger, A. Santoni, F. Leung, D. Rodgers, and R. Uhlig, "Intel virtualization technology: Hardware support for efficient processor virtualization.," *Intel Technology Journal*, vol. 10, no. 3, 2006.

[5] M. Bazm, R. Khatoun, Y. Begriche, L. Khoukhi, X. Chen, and A. Serhrouchni, "Malicious virtual machines detection through a clustering approach," in *Cloud Technologies and Applications (CloudTech), 2015 International Conference on*, pp. 1–8, IEEE, 2015.

[6] J. Fan, X. Guo, E. De Mulder, P. Schaumont, B. Preneel, and I. Verbauwhede, "State-of-the-art of secure ecc implementations: A survey on known side-channel attacks and countermeasures.," *HOST*, vol. 2010, pp. 76–87, 2010.

[7] Y. Zhou and D. Feng, "Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing.," *IACR Cryptology ePrint Archive*, vol. 2005, p. 388, 2005.

[8] A. Litchfield and A. Shahzad, "Virtualization technology: Cross-vm cache side channel attacks make it vulnerable," *arXiv preprint arXiv:1606.01356*, 2016.

[9] V. Varadarajan, T. Ristenpart, and M. Swift, "Scheduler-based defenses against cross-vm side-channels," in *23rd USENIX Security Symposium (USENIX Security 14)*, pp. 687–702, 2014.

[10] J. Xiao, Z. Xu, H. Huang, and H. Wang, "Security implications of memory deduplication in a virtualized environment," in *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 1–12, IEEE, 2013.

[11] Y. Zheng, B. T. Davis, and M. Jordan, "Performance evaluation of exclusive cache hierarchies," in *Performance Analysis of Systems and Software, 2004 IEEE International Symposium on-ISPASS*, pp. 89–96, IEEE, 2004.

[12] "Cortex-A Series, Programmers Guide, http://www.csc.lsu.edu/~whaley/teach/fhpo_f11/arm/cortaprogguide.pdf."

[13] M. E. Thomadakis, "The architecture of the nehalem processor and nehalem-ep smp platforms," *Resource*, vol. 3, p. 2, 2011.

[14] J. Demme, R. Martin, A. Waksman, and S. Sethumadhavan, "Side-channel vulnerability factor: a metric for measuring information leakage," *ACM SIGARCH Computer Architecture News*, vol. 40, no. 3, pp. 106–117, 2012.

[15] C. Percival, "Cache missing for fun and profit," 2005.

[16] D. Harnik, B. Pinkas, and A. Shulman-Peleg, "Side channels in cloud services: Deduplication in cloud storage," *IEEE Security & Privacy*, vol. 8, no. 6, pp. 40–47, 2010.

[17] A. Arcangeli, I. Eidus, and C. Wright, "Increasing memory density by using ksm," in *Proceedings of the linux symposium*, pp. 19–28, Citeseer, 2009.

[18] K. Suzaki, K. Iijima, T. Yagi, and C. Artho, "Memory deduplication as a threat to the guest os," in *Proceedings of the Fourth European Workshop on System Security*, p. 1, ACM, 2011.

[19] E. Bosman, K. Razavi, H. Bos, and C. Giuffrida, "Dedup est machina: Memory deduplication as an advanced exploitation vector," 2016.

[20] D. Gruss, D. Bidner, and S. Mangard, "Practical memory deduplication attacks in sandboxed javascript," in *European Symposium on Research in Computer Security*, pp. 108–122, Springer, 2015.

[21] J. Xiao, Z. Xu, H. Huang, and H. Wang, "A covert channel construction in a virtualized environment," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, (New York, NY, USA), pp. 1040–1042, ACM, 2012.

[22] G. Irazoqui, T. Eisenbarth, and B. Sunar, "Jackpot stealing information from large caches via huge pages,"

[23] V. Costan and S. Devadas, "Intel sgx explained," tech. rep., Cryptology ePrint Archive, Report 2016/086, 2016. https://eprint. iacr. org/2016/086.

[24] A. Parashar and R. Arora, "Secure user data in cloud computing using encryption algorithms," *International journal of engineering research and applications*, vol. 3, pp. 1922–1926, 2013.

[25] H. Li, Y. Dai, L. Tian, and H. Yang, "Identity-based authentication for cloud computing," in *IEEE International Conference on Cloud Computing*, pp. 157–166, Springer, 2009.

[26] M. Weiß, B. Heinz, and F. Stumpf, "A cache timing attack on aes in virtualization environments," in *International Conference on Financial Cryptography and Data Security*, pp. 314–328, Springer, 2012.

[27] K. Okamura and Y. Oyama, "Load-based covert channels between xen virtual machines," in *Proceedings of the 2010 ACM Symposium on Applied Computing*, pp. 173–180, ACM, 2010.

[28] P. Conway, N. Kalyanasundharam, G. Donley, K. Lepak, and B. Hughes, "Cache hierarchy and memory subsystem of the amd opteron processor," *IEEE micro*, vol. 30, no. 2, pp. 16–29, 2010.

[29] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM conference on Computer and communications security*, pp. 199–212, ACM, 2009.

[30] G. Irazoqui, T. Eisenbarth, and B. Sunar, "S$a: A shared cache attack that works across cores and defies vm sandboxing–and its application to aes," in *2015 IEEE Symposium on Security and Privacy*, pp. 591–604, IEEE, 2015.

[31] M. S. Inci, B. Gulmezoglu, G. Irazoqui, T. Eisenbarth, and B. Sunar, "Seriously, get off my cloud! cross-vm rsa key recovery in a public cloud," tech. rep., Cryptology ePrint Archive, Report 2015/898, 2015. http://eprint. iacr. org, 2015.

[32] D. Gullasch, E. Bangerter, and S. Krenn, "Cache games–bringing access-based cache attacks on aes to practice," in *2011 IEEE Symposium on Security and Privacy*, pp. 490–505, IEEE, 2011.

[33] Y. Oren, V. P. Kemerlis, S. Sethumadhavan, and A. D. Keromytis, "The spy in the sandbox–practical cache attacks in javascript," *arXiv preprint arXiv:1502.07373*, 2015.

[34] Y. Yarom and K. Falkner, "Flush+ reload: a high resolution, low noise, l3 cache side-channel attack," in *23rd USENIX Security Symposium (USENIX Security 14)*, pp. 719–732, 2014.

[35] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *IEEE Symposium on Security and Privacy*, pp. 605–622, 2015.

[36] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+ flush: A fast and stealthy cache attack," *arXiv preprint arXiv:1511.04594*, 2015.

[37] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-tenant side-channel attacks in paas clouds," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pp. 990–1003, ACM, 2014.

[38] R. Spreitzer and T. Plos, "Cache-access pattern attack on disaligned aes t-tables," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pp. 200–214, Springer, 2013.

[39] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Annual International Cryptology Conference*, pp. 104–113, Springer, 1996.

[40] J.-F. Dhem, F. Koeune, P.-A. Leroux, P. Mestré, J.-J. Quisquater, and J.-L. Willems, "A practical implementation of the timing attack," in *International Conference on Smart Card Research and Advanced Applications*, pp. 167–182, Springer, 1998.

[41] D. Brumley and D. Boneh, "Remote timing attacks are practical," *Computer Networks*, vol. 48, no. 5, pp. 701–716, 2005.

[42] O. Aciiçmez, W. Schindler, and c. K. Koç, "Improving brumley and boneh timing attack on unprotected ssl implementations," in *Proceedings of the 12th ACM Conference on Computer and Communications Security*, CCS '05, (New York, NY, USA), pp. 139–146, ACM, 2005.

[43] J. Bonneau and I. Mironov, "Cache-collision timing attacks against aes," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 201–215, Springer, 2006.

[44] D. J. Bernstein, "Cache-timing attacks on aes," 2005.

[45] O. Acıiçmez, W. Schindler, and Ç. K. Koç, "Cache based remote timing attack on the aes," in *Cryptographers Track at the RSA Conference*, pp. 271–286, Springer, 2007.

[46] Y. Tsunoo, T. Saito, T. Suzaki, M. Shigeri, and H. Miyauchi, "Cryptanalysis of des implemented on computers with cache," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 62–76, Springer, 2003.

[47] O. Acıiçmez and Ç. K. Koç, "Trace-driven cache attacks on aes (short paper)," in *International Conference on Information and Communications Security*, pp. 112–121, Springer, 2006.

[48] J.-F. Gallais, I. Kizhvatov, and M. Tunstall, "Improved trace-driven cache-collision attacks against embedded aes implementations," in *International Workshop on Information Security Applications*, pp. 243–257, Springer, 2010.

[49] G. Irazoqui, M. S. Inci, T. Eisenbarth, and B. Sunar, "Wait a minute! a fast, cross-vm attack on aes," in *International Workshop on Recent Advances in Intrusion Detection*, pp. 299–319, Springer, 2014.

[50] B. Gülmezoğlu, M. S. Inci, G. Irazoqui, T. Eisenbarth, and B. Sunar, "A faster and more realistic flush+ reload attack on aes," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pp. 111–126, Springer, 2015.

[51] N. Benger, J. van de Pol, N. P. Smart, and Y. Yarom, "ooh aah... just a little bit: A small amount of side channel can go a long way," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 75–92, Springer, 2014.

[52] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: the case of aes," in *Cryptographers Track at the RSA Conference*, pp. 1–20, Springer, 2006.

[53] D. Gruss, R. Spreitzer, and S. Mangard, "Cache template attacks: Automating attacks on inclusive last-level caches," in *24th USENIX Security Symposium (USENIX Security 15)*, (Washington, D.C.), pp. 897–912, USENIX Association, 2015.

[54] C. Disselkoen, D. Kohlbrenner, L. Porter, and D. Tullsen, "Prime+abort: A timer-free high-precision l3 cache attack using intel TSX," in *26th USENIX Security Symposium (USENIX Security 17)*, (Vancouver, BC), pp. 51–67, USENIX Association, 2017.

[55] V. Varadarajan, Y. Zhang, T. Ristenpart, and M. M. Swift, "A placement vulnerability study in multi-tenant public clouds.," in *USENIX Security*, pp. 913–928, 2015.

[56] Y. Yarom, Q. Ge, F. Liu, R. B. Lee, and G. Heiser, "Mapping the intel last-level cache," tech. rep., IACR Cryptology ePrint Archive, Report 2015/905, 2015.

[57] C. Maurice, N. Le Scouarnec, C. Neumann, O. Heen, and A. Francillon, "Reverse engineering intel last-level cache complex addressing using performance counters," in *International Workshop on Recent Advances in Intrusion Detection*, pp. 48–65, Springer, 2015.

[58] G. Irazoqui, T. Eisenbarth, and B. Sunar, "Systematic reverse engineering of cache slice selection in intel processors," in *Digital System Design (DSD), 2015 Euromicro Conference on*, pp. 629–636, IEEE, 2015.

[59] P. Pessl, D. Gruss, C. Maurice, M. Schwarz, and S. Mangard, "Reverse engineering intel dram addressing and exploitation," *arXiv preprint arXiv:1511.08756*, 2015.

[60] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-vm side channels and their use to extract private keys," in *Proceedings of the 2012 ACM conference on Computer and communications security*, pp. 305–316, ACM, 2012.

[61] V. Gopal, J. Guilford, E. Ozturk, W. Feghali, G. Wolrich, and M. Dixon, "Fast and constant-time implementation of modular exponentiation," in *28th International Symposium on Reliable Distributed Systems. Niagara Falls, New York, USA*, 2009.

[62] T. Izu and T. Takagi, "A fast parallel elliptic curve multiplication resistant against side channel attacks," in *International Workshop on Public Key Cryptography*, pp. 280–296, Springer, 2002.

[63] B. Möller, "Securing elliptic curve point multiplication against side-channel attacks," in *International Conference on Information Security*, pp. 324–334, Springer, 2001.

[64] M. Hamburg, "Accelerating aes with vector permute instructions," in *Cryptographic Hardware and Embedded Systems-CHES 2009*, pp. 18–32, Springer, 2009.

[65] J. Blömer, J. Guajardo, and V. Krummel, "Provably secure masking of aes," in *International Workshop on Selected Areas in Cryptography*, pp. 69–83, Springer, 2004.

[66] M. Rivain and E. Prouff, "Provably secure higher-order masking of aes," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 413–427, Springer, 2010.

[67] H. Kim, S. Hong, and J. Lim, "A fast and provably secure higher-order masking of aes s-box," in *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 95–107, Springer, 2011.

[68] A. Askarov, D. Zhang, and A. C. Myers, "Predictive black-box mitigation of timing channels," in *Proceedings of the 17th ACM conference on Computer and communications security*, pp. 297–307, ACM, 2010.

[69] G. Barthe, T. Rezk, and M. Warnier, "Preventing timing leaks through transactional branching instructions," *Electronic Notes in Theoretical Computer Science*, vol. 153, no. 2, pp. 33–55, 2006.

[70] J. V. Cleemput, B. Coppens, and B. De Sutter, "Compiler mitigations for time attacks on modern x86 processors," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 8, no. 4, p. 23, 2012.

[71] B. Coppens, I. Verbauwhede, K. De Bosschere, and B. De Sutter, "Practical mitigations for timing-based side-channel attacks on modern x86 processors," in *2009 30th IEEE Symposium on Security and Privacy*, pp. 45–60, IEEE, 2009.

[72] J. Kelsey, B. Schneier, D. Wagner, and C. Hall, "Side channel cryptanalysis of product ciphers," in *European Symposium on Research in Computer Security*, pp. 97–110, Springer, 1998.

[73] E. Brickell, G. Graunke, M. Neve, and J.-P. Seifert, "Software mitigations to hedge aes against cache-based software side channel vulnerabilities.," *IACR Cryptology ePrint Archive*, vol. 2006, p. 52, 2006.

[74] C. Liu, A. Harris, M. Maas, M. Hicks, M. Tiwari, and E. Shi, "Ghostrider: A hardware-software system for memory trace oblivious computation," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 1, pp. 87–101, 2015.

[75] C. Liu, M. Hicks, and E. Shi, "Memory trace oblivious program execution," in *2013 IEEE 26th Computer Security Foundations Symposium*, pp. 51–65, IEEE, 2013.

[76] Y. Zhang and M. K. Reiter, "Düppel: retrofitting commodity operating systems to mitigate cache side channels in the cloud," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pp. 827–838, ACM, 2013.

[77] M. Godfrey and M. Zulkernine, "A server-side solution to cache-based side-channel attacks in the cloud," in *2013 IEEE Sixth International Conference on Cloud Computing*, pp. 163–170, IEEE, 2013.

[78] B. A. Braun, S. Jana, and D. Boneh, "Robust and efficient elimination of cache and timing side channels," *CoRR*, vol. abs/1506.00189, 2015.

[79] D. Stefan, P. Buiras, E. Z. Yang, A. Levy, D. Terei, A. Russo, and D. Mazières, "Eliminating Cache-Based Timing Attacks with Instruction-Based Scheduling," *Esorics*, vol. 8134, pp. 718–735, 2013.

[80] S.-J. Moon, V. Sekar, and M. K. Reiter, "Nomad: Mitigating arbitrary cloud side channels via provider-assisted migration," in *Proceedings of the 22nd acm sigsac conference on computer and communications security*, pp. 1595–1606, ACM, 2015.

[81] R. Zhuang, S. A. DeLoach, and X. Ou, "Towards a theory of moving target defense," in *Proceedings of the First ACM Workshop on Moving Target Defense*, pp. 31–40, ACM, 2014.

[82] Y. Zhang, M. Li, K. Bai, M. Yu, and W. Zang, "Incentive compatible moving target defense against vm-colocation attacks in clouds," in *IFIP International Information Security Conference*, pp. 388–399, Springer, 2012.

[83] "Eliminating fine grained timers in Xen," *ACM Workshop on Cloud Computing Security*, p. 41, 2011.

[84] X. Jin, H. Chen, X. Wang, Z. Wang, X. Wen, Y. Luo, and X. Li, "A simple cache partitioning approach in a virtualized environment," in *2009 IEEE International Symposium on Parallel and Distributed Processing with Applications*, pp. 519–524, IEEE, 2009.

[85] D. Tam, R. Azimi, L. Soares, and M. Stumm, "Managing shared l2 caches on multicore systems in software," in *Workshop on the Interaction between Operating Systems and Computer Architecture*, pp. 26–33, Citeseer, 2007.

[86] J. Shi, X. Song, H. Chen, and B. Zang, "Limiting cache-based side-channel in multi-tenant cloud using dynamic page coloring," in *2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W)*, pp. 194–199, IEEE, 2011.

[87] H. Raj, R. Nathuji, A. Singh, and P. England, "Resource management for isolation enhanced cloud services," in *Proceedings of the 2009 ACM workshop on Cloud computing security*, pp. 77–84, ACM, 2009.

[88] T. Kim, M. Peinado, and G. Mainar-Ruiz, "Stealthmem: system-level protection against cache-based side channel attacks in the cloud," in *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, pp. 189–204, 2012.

[89] J. Kong, O. Aciicmez, J.-P. Seifert, and H. Zhou, "Deconstructing new cache designs for thwarting software cache-based side channel attacks," in *Proceedings of the 2nd ACM workshop on Computer security architectures*, pp. 25–34, ACM, 2008.

[90] J. Kong, O. Aciiçmez, J.-P. Seifert, and H. Zhou, "Hardware-software integrated approaches to defend against software cache-based side channel attacks," in *2009 IEEE 15th International Symposium on High Performance Computer Architecture*, pp. 393–404, IEEE, 2009.

[91] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," in *ACM SIGARCH Computer Architecture News*, vol. 35, pp. 494–505, ACM, 2007.

[92] Z. Wang and R. B. Lee, "A novel cache architecture with enhanced performance and security," in *2008 41st IEEE/ACM International Symposium on Microarchitecture*, pp. 83–93, IEEE, 2008.

[93] D. Page, "Partitioned cache architecture as a side-channel defence mechanism.," *IACR Cryptology ePrint Archive*, vol. 2005, p. 280, 2005.

[94] L. Domnitser, A. Jaleel, J. Loew, N. Abu-Ghazaleh, and D. Ponomarev, "Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 8, no. 4, p. 35, 2012.

[95] "Intel Microarchitecture Codename Nehalem Performance Monitoring Unit Programming Guide , https://software.intel.com/sites/default/files/m/5/2/c/f/1/30320-nehalem-pmu-programming-guide-core.pdf."

[96] T. Zhang, Y. Zhang, and R. B. Lee, "Cloudradar: A real-time side-channel attack detection system in clouds," in *International Symposium on Research in Attacks, Intrusions, and Defenses*, pp. 118–140, Springer, 2016.

[97] "Improve real-time performance utilizing cache allocation technology," tech. rep., Intel Corporation, April 2015.

[98] F. Liu, Q. Ge, Y. Yarom, F. Mckeen, C. Rozas, G. Heiser, and R. B. Lee, "Catalyst: Defeating last-level cache side channel attacks in cloud computing," in *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pp. 406–418, IEEE, 2016.

[99] "Securing the Enterprise with Intel AES-NI, http://www.intel.com/content/dam/doc/white-paper/enterprise-security-aes-ni-white-paper.pdf."

[100] T. Zhang, Y. Zhang, and R. B. Lee, "Memory dos attacks in multi-tenant clouds: Severity and mitigation," *CoRR*, vol. abs/1603.03404, 2016.

[101] O. Goldreich, "Secure multi-party computation," *Manuscript. Preliminary version*, pp. 86–97, 1998.

[102] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa, "Oblivious multi-party machine learning on trusted processors," in *USENIX Security*, vol. 16, pp. 619–636, 2016.

[103] A. Lebre, J. Pastor, D. Consortium, *et al.*, *The DISCOVERY Initiative-Overcoming Major Limitations of Traditional Server-Centric Clouds by Operating Massively Distributed IaaS Facilities*. PhD thesis, Inria Rennes Bretagne Atlantique, 2015.

[104] M. Schwarz, S. Weiser, D. Gruss, C. Maurice, and S. Mangard, "Malware guard extension: Using sgx to conceal cache attacks," *arXiv preprint arXiv:1702.08719*, 2017.